Getting started

This chapter assumes you've already installed Ansible-cmdb.

First, generate Ansible output for your hosts:

```
mkdir out
ansible -m setup --tree out/ all
```

Next, call ansible-cmdb on the resulting `out/` directory to generate the CMDB overview page:

```
ansible-cmdb out/ > overview.html
```

By default, the `html_fancy` template is used, which generates output containing an overview of all your hosts, with a section of detailed information for each host.

You can now open `overview.html` in your browser to view the results.

Full usage

```
Usage: ansible-cmdb [option] <dir> > output.html

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -t TEMPLATE, --template=TEMPLATE
                        Template to use. Default is 'html_fancy'
  -i INVENTORY, --inventory=INVENTORY
                        Inventory to read extra info from
  -f, --fact-cache      <dir> contains fact-cache files
  -p PARAMS, --params=PARAMS
                        Params to send to template
  -d, --debug           Show debug output
  -q, --quiet           Don't report warnings
  -c COLUMNS, --columns=COLUMNS
                        Show only given columns
  -C CUST_COLS, --cust-cols=CUST_COLS
                        Path to a custom columns definition file
  -l LIMIT, --limit=LIMIT
                        Limit hosts to pattern
  --exclude-cols=EXCLUDE_COLUMNS
                        Exclude cols from output
```

Inventory scanning

Ansible-cmdb can read your inventory file (`hosts`, by default), inventory directory or dynamic inventory and extract useful information from it such as:

- All the groups a host belongs to.
- Host variables. These are optional key/value pairs for each host which can be used in playbooks. They are scanned by ansible-cmdb and get added to a hosts discovered facts under the 'hostvars' section.

Reading the inventory is done using the `-i` switch to ansible-cmdb. It takes a single parameter: your hosts file, directory containing your hosts files or path to your dynamic inventory script. You may specify multiple inventory files by separating them with a comma (do not include spaces!).

For example:

```
$ ansible-cmdb -i ./hosts out/ > overview.html
```

If `host_vars` and `group_vars` directories exist at that location, they will be included automatically.

Host and group variables

Ansible-cmdb reads and includes the host and group variables from the inventory. When you point ansible-cmdb to your host inventory (`hosts` file, usually) with the `-i` option, ansible-cmdb automatically includes information from the `host_vars` and `group_vars` directories if found in the same dir.

Whether the variables are included in the output depends on the template used. In the `html_fancy` templates, variables will become available under the "Custom variables" heading.

The ''html_fancy'' template supports four special variables:

- `groups`: A list of Ansible groups the host belongs to.
- `dtap`: Whether a host is a development, test, acceptance or production system.
- `comment`: A comment for the host.
- `ext_id`: An external unique identifier for the host.

For example, let's say we have the following `hosts` file:

v: latest

```
[cust.megacorp]
db1.dev.megacorp.com   dtap=dev  comment="Old database server"
db2.dev.megacorp.com   dtap=dev  comment="New database server"
test.megacorp.com      dtap=test
acc.megacorp.com       dtap=acc  comment="24/7 support"
megacorp.com           dtap=prod comment="Hosting by Foo" ext_id="SRV_10029"

[os.redhat]
megacorp.com
acc.megacorp.com
test.megacorp.com
db2.dev.megacorp.com

[os.debian]
db1.dev.megacorp.com
```

The host `acc.megacorp.com` will have groups 'cust.megacorp' and 'os.redhat', will have a comment saying it has 24/7 support and will be marked as a `acc` server. Megacorp.com host will have an external ID of "SRV_10029", which will be required by for communicating with Foo company about hosting.

These variables are included in the host overview table in the `html_fancy` templates.

See the official Ansible documentation on Host variables for more information on host variables.

Specifying templates

ansible-cmdb offers multiple templates. You can choose your template with the `-t` or `--template` argument:

```
ansible-cmdb -t tpl_custom out/ > overview.html
```

The 'html_fancy' template is the default.

Templates can be referred to by name or by relative/absolute path to the `.tpl` file. This lets you implement your own templates. For example:

```
$ ansible-cmdb -t /home/fboender/my_template out/ > my_template.html
```

Template parameters

Some templates support parameters that influence their output. Parameters are specified using the `-p` or `--parameter` option to `ansible-cmdb`. Multiple parameters may be specified by separating them with commas. There must be *no* spaces in the parameters.

For example, to specify the `html_fancy` template with local Javascript libraries and closed trees:

```
ansible-cmdb -t html_fancy -p local_js=1,collapsed=1 out > overview.html
```

Available templates

Ansible-cmdb currently provides the following templates out of the box:

- `html_fancy`: A dynamic, modern HTML page containing all hosts.
- `html_fancy_split`: A dynamic, modern HTML page with each host's details in a separate file.
- `txt_table`: A quick text table summary of the available hosts with some minimal information.
- `json`: Dumps all hosts including groups, variable, custom info in JSON format.
- `csv`: The CSV template outputs a CSV file of your hosts.
- `markdown`: The Markdown template generates host information in the Markdown format.
- `sql`: The SQL template generates an .sql file that can be loaded into an SQLite or MySQL database.

html_fancy:

`html_fancy` is currently the default template.

A fancy HTML page that uses jQuery and DataTables to give you a searchable, sortable table overview of all hosts with detailed information just a click away.

It takes optional parameters:

- `local_js=0|1`: Load resources from local disk (default=`0`). If set, will load resources from the local disk instead of over the network.
- `collapsed=0|1`: Controls whether host information is collapsed by default or not. A value of `1` will collapse all host information by default (default='0').
- `host_details=0|1`: Render host details or not. (default=`1`)
- `skip_empty=0|1`: Skip hosts for which no facts were gathered (unreachable, etc). (default=`0`).

html_fancy_split:

This template is basically the same as the **html_fancy** template, but it generates a `cmdb/` directory with an `index.html` file and a separate html file for each host's details. This is useful if you have a large amount of hosts and the html_fancy template is rendering too slow.

Usage:

```
ansible-cmdb -t html_fancy_split -i hosts out/
```

It accepts the same parameters as the `html_fancy` template.

sql:

The `sql` template generates an .sql file that can be loaded into an SQLite or MySQL database.

```
$ ansible-cmdb -t sql -i hosts out > cmdb.sql
$ echo "CREATE DATABASE ansiblecmdb" | mysql
$ mysql ansiblecmdb < cmdb.sql
```

Fact caching

Ansible can cache facts from hosts when running playbooks. This is configured in Ansible like:

```
[defaults]
fact_caching=jsonfile
fact_caching_connection = /path/to/facts/dir
```

You can use these cached facts as facts directories with ansible-cmdb by specifying the `-f` (`--fact-cache`) option:

```
$ ansible-cmdb -f /path/to/facts/dir > overview.html
```

Please note that the `--fact-cache` option will apply to *all* fact directories you specify. This means you can't mix fact-cache fact directories and normal `setup` fact directories. Also, if you wish to manually extend facts (see the `Extending` chapter), you must omit the `ansible_facts` key and put items in the root of the JSON.

Limiting hosts

You can use the `-l` parameter to limit the hosts included in the output. This works basically the same as Ansible's limits. It supports the '`all`' construct, individual hosts, groups and inversion using `!`. It does **not** currently support wildcards (e.g. `db*.example.com`) or host expansions (e.g. `db[0-3].example.com`) In order for host limiting to work, you must point the `-i` param to your host inventory.

Some examples:

```
# Include only hosts in the group 'cust.acme'
ansible-cmdb -i hosts -l 'cust.acme' out/ > cmdb.html

# Include all hosts except those in the group 'cust.acme', but include
# host 'foo.example.com'
ansible-cmdb -i hosts -l 'all:!cust.acme:foo.example.com' out/ > cmdb.html
```

Columns

Some templates, such as txt_table and html_fancy, support columns. If a template supports columns, you can use the `--columns` / `-c` command line option to specify which columns to show.

The `--columns` takes a comma separated list of columns (no spaces!) which should be shown. The columns must be specified by their `id` field. For information on what `id` fields are supported by a template, take a look in the template. Usually it's the column title, but in lowercase and with spaces replaced by underscores.

For example:

```
$ ansible-cmdb -t txt_table --columns name,os,ip,mem,cpus facts/
Name                     OS             IP             Mem  CPUs
----------------------   -------------  -------------  ---  -
jib.electricmonk.nl      Linuxmint 17   192.168.0.3    16g  1
app.uat.local            Debian 6.0.10  192.168.57.1   1g   1
eek.electricmonk.nl      Ubuntu 14.04   192.168.0.10   3g   1
db01.prod.local          Debian 6.0.10  192.168.58.1   0g   1
debian.dev.local         Debian 6.0.10  192.168.56.2   1g   1
db02.prod.local          Debian 6.0.10  192.168.58.2   0g   1
centos.dev.local         CentOS 6.6     192.168.56.8   1g   1
win.dev.local            Windows 2012   10.0.0.3       4g   0
host5.example.com        Debian 6.0.10  192.168.57.1   1g   1
db03.prod.local          Debian 6.0.10  192.168.58.3   0g   1
zoltar.electricmonk.nl   Ubuntu 14.04   194.187.79.11  4g   2
```

You can use the `--exclude-cols` option to exclude specific columns. It works the same as `--columns`. For example:

```
ansible-cmdb -t html_fancy_split \
             --exclude-cols mem_usage,swap_usage,disk_usage,physdisk_size \
             -i hosts \
             facts/
```

If you want to add custom columns, please refer to Custom columns section.

Extending facts

You can specify multiple directories that need to be scanned for facts. This lets you overri v: latest extend and fill in missing information on hosts. You can also use this to create completely new hosts or to add custom facts to your hosts.

Extended facts are basically the same as normal Ansible fact files. When you specify multiple fact directories, Ansible-cmdb scans all of the in order and overlays the facts.

Note that the host *must still* be present in your hosts file, or it will not generate anything.

If you're using the `--fact-cache` option, you must omit the `ansible_facts` key and put items in the root of the JSON. This also means that you can only extend native ansible facts and not information read from the `hosts` file by ansible-cmdb.

Override and fill in facts

Sometimes Ansible doesn't properly gather certain facts for hosts. For instance, OpenBSD facts don't include the `userspace_architecture` fact. You can add it manually to a host.

Create a directory for your extended facts:

```
$ mkdir out_extend
```

Create a file in it for a host. The file must be named the same as it appears in your `hosts` file:

```
$ vi out_extend/openbsd.dev.local
{
  "ansible_facts": {
      "ansible_userspace_architecture": "x86_64"
  }
}
```

Specify both directories when generating the output:

```
./ansible-cmdb out/ out_extend/ > overview.html
```

Your OpenBSD host will now include the 'Userspace Architecture' fact.

Manual hosts

For example, lets say you have 100 linux machines, but only one windows machine. It's not worth setting up ansible on that one windows machine, but you still want it to appear in your overview...

Create a directory for you custom facts:

```
$ mkdir out_manual
```

Create a file in it for your windows host:

```
$ vi out_manual/win.dev.local
{
  "groups": [
  ],
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "10.10.0.2",
      "191.37.104.122"
    ],
    "ansible_default_ipv4": {
      "address": "191.37.104.122"
    },
    "ansible_devices": {
    },
    "ansible_distribution": "Windows",
    "ansible_distribution_major_version": "2008",
    "ansible_distribution_release": "",
    "ansible_distribution_version": "2008",
    "ansible_domain": "win.dev.local",
    "ansible_fips": false,
    "ansible_form_factor": "VPS",
    "ansible_fqdn": "win.dev.local",
    "ansible_hostname": "win",
    "ansible_machine": "x86_64",
    "ansible_nodename": "win.dev.local",
    "ansible_userspace_architecture": "x86_64",
    "ansible_userspace_bits": "64",
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "xen",
    "module_setup": true
  },
  "changed": false
}
```

Now you can create the overview including the windows host by specifying two fact directories:

```
./ansible-cmdb out/ out_manual/ > overview.html
```

Custom facts

You can add custom facts (not to be confused with 'custom variables') to you hosts. These facts will be displayed in the `html_fancy` template by default under the 'Custom facts' header.

Note that these are not the same as Host local facts. Host local facts are facts that Ansible reads from each of your host's `/etc/ansible/facts.d` directory. Those are also included in Ansible-cmdb's html_fancy templates, but under the "Host local facts" heading. The custom facts explained here are manually defined on the host where you run ansible-cmdb, and have little to do with Ansible itself.

Let's say you want to add information about installed software to your facts.

Create a directory for you custom facts:

```
$ mkdir out_custom
```

Create a file in it for the host where you want to add the custom facts:

```
$ vi custfact.test.local
{
  "custom_facts": {
    "software": {
      "apache": {
        "version": "2.4",
        "install_src": "backport_deb"
      },
      "mysql-server": {
        "version": "5.5",
        "install_src": "manual_compile"
      },
      "redis": {
        "version": "3.0.7",
        "install_src": "manual_compile"
      }
    }
  }
}
```

For this to work the facts **must** be listed under the **custom_facts** key.

Generate the overview:

```
./ansible-cmdb out/ out_custom/ > overview.html
```

The software items will be listed under the "*Custom facts*" heading.

Custom columns

You can add custom columns to the host overview with the `-C` (`--cust-cols`) option. This allows you to specify jsonxs expressions or Mako template fragments to extract and display custom host facts.

Custom columns are currently only supported by the `html_fancy` and `html_fancy_split` templates.

The `-C` option takes a parameter which is the path to a file containing your custom column definitions. The file's syntax is Python (even though it looks like JSON). An example can be found in the `examples/cust_cols.conf` file in the repo:

```
[
    # Show whether AppArmor is enabled
    {
        "title": "AppArmor",
        "id": "apparmor",
        "sType": "string",
        "visible": False,
        "jsonxs": "ansible_facts.ansible_apparmor.status"
    },
    # Show the nameservers configured on the host
    {
        "title": "Nameservers",
        "id": "nameservers",
        "sType": "string",
        "visible": True,
        "tpl": """
          <ul>
            <%
            # Get ansible_facts.ansible_dns.nameservers
            facts = host.get('ansible_facts', {})
            dns = facts.get('ansible_dns', {})
            nameservers = dns.get('nameservers', [])
            %>
            % for nameserver in nameservers:
              <li>${nameserver}</li>
            % endfor
          </ul>
        """
    },
    # Show the nameservers configured on the host, but use jsonxs.
    {
        "title": "Nameservers2",
        "id": "nameservers2",
        "sType": "string",
        "visible": True,
        "tpl": """
          <ul>
            <%
            # Get ansible_facts.ansible_dns.nameservers using jsonxs
            nameservers = jsonxs(host, 'ansible_facts.ansible_dns.nameservers', default=[])
            %>
            % for nameserver in nameservers:
              <li>${nameserver}</li>
            % endfor
          </ul>
        """
    }
]
```

This defines two new columns: 'AppArmor' and 'Nameservers'. Each column consist of the following key/values:

- `title` is what is displayed as the columns user-friendly title. **Required**.
- The `id` key must have a unique value, to differentiate between columns. **Required**
- The `sType` value determines how the values will be sorted in the host overview. Possible values include `string` and `num`. **Required**
- `visible` determines whether the column will be active (shown) by default. **Required**
- The `jsonxs` expression, if specified points to an entry in the facts files for each host, and determines what will be shown for the column's value for each host. The easiest way tc v: latest out a jsonxs expression is by opening one of the gathered facts files in a json editor. Please see jsonxs for info on how to write jsonxs expressions. **Optional**

- The `tpl` expression, if specified, is a Mako template fragment. A single variable `host` is made available in this template. Care must be taken when accessing host information. If one of the hosts is missing the information you're trying to access, the template will not render and ansible-cmdb will crash (usually with a 'KeyError' message). You should always use the `get()` method and specify a default value. E.g. `host.get('ansible_facts', {}).get('ansible_dns', {}).get('nameservers', [])`. Alternatively (and recommended) is that you use `jsonxs` to access your info (and specify `default=...`). See the example above. **Optional**

Either `jsonxs` or `tpl` is required.

To use it:

```
ansible-cmdb -C example/cust_cols.conf -i example/hosts example/out/ > cmdb.html
```

When opening the `cmdb.html` file in your browser, you may have to hit the 'Clear settings' button in the top-right before the new columns show up or when you get strange behaviour.

Custom templates

It's possible to create custom templates to build completely different CMDBs or to enhance the existing ones. Ansible-cmdb uses the Mako templating engine to render output.

For example, if you want to add a custom column to the `html_fancy` template (note that it's easier to just use the `--cust-cols` option. For more info see above):

1. Make a copy of the default `html_fancy` template in a new dir. Here, we'll use files from the ansible-cmdb git repository.

```
$ mkdir ~/mytemplate
$ cp ~/ansible-cmdb/src/ansiblecmdb/data/tpl/html_fancy.tpl ~/mytemplate/
$ cp ~/ansible-cmdb/src/ansiblecmdb/data/tpl/html_fancy_defs.html ~/mytemplate/
```

2. Edit the `html_fancy_defs.html` file and add an entry to the `cols =` section. In this example, we'll add a column for the "BIOS version".

v: latest

```
<%
  cols = [
   ...
   {"title": "Product Serial","id": "prodserial",    "func": col_prodserial,     "sType": "string
   {"title": "BIOS version",  "id": "bios_version",  "func": col_bios_version,   "sType": "string
 ]
```

3. Now you need to implement the `col_biosversion` template function. In the same `html_fancy_defs.html` file, search for the `## Column functions` section. Add a column template function called `col_biosversion`:

```
   <%def name="col_dtap(host, **kwargs)">
     ${jsonxs(host, 'hostvars.dtap', default='')}
   </%def>

+  <%def name="col_bios_version(host, **kwargs)">
+    ${jsonxs(host, 'ansible_facts.ansible_bios_version', default='')}
+  </%def>
```

4. Finally, render the custom template. For this to work, you **must be in the same directory as the custom template!**.

```
ansible-cmdb/src/ansible-cmdb -t ./html_fancy -i ~/ansible/hosts ~/ansible/out/ > cmdb.html
```

v: latest